FEATURE ARTICLE

Robert Schreiber

Levitating a Beach Ball Using Fuzzy Logic

Wanting a hot trade show demo, Microchip takes up Tom Cantrell's PID-pong challenge. Not only do they get a beach ball hovering near the top of a large plastic tube, they do it all with fuzzy logic.

When we took on the task of coming up with a project for a recent trade show, we were inspired by the PID-Pong demo described by Tom Cantrell's "Silicon Update" (*INK* 42, 50). Tom's demo used a PID algorithm to set the fan speed. In turn, the fan controlled the height of a ping-pong ball in a vertical tube.

However, ping-pong did not fit the trade show's "Beach Party" theme. So, we upped the ante, replacing the ping-pong ball with a beach ball.

DEMO DESCRIPTION

The trade show demo we came up with is shown in Figure 1. A control panel prompts the user to enter the desired beach ball height on the 16-key keypad. The keypad input echoes on the LCD module and the user is prompted for confirmation.

On confirmation of user input, the control panel initiates a ranging cycle to calculate the current height of the beach ball. The desired height and current height are continually displayed on the LCD module. From the current height, the control panel calculates both the velocity and the delta height (i.e., difference in desired height from current height).

This information, along with the desired height, is transmitted to the PC via an RS-232 link. The fuzzy logic algorithm, running on the PC, calculates the appropriate duty cycle of the DC fan and transmits this information to the control panel. This emulates a real-world environment in which system-level debugging can be done on the PC in real-time. The control panel controls the duty cycle of the DC fan with this input.

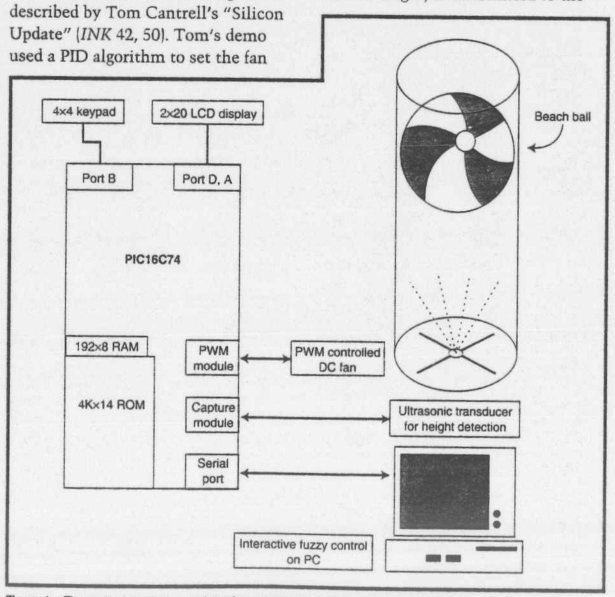


Figure 1—*The trade show demo consists of a control panel and demo cabinet. The demo cabinet contains the DC fan, transducer, power supplies, and the 6' clear tube. The control panel houses the PIC16C74 microcontroller, which provides the "brains" for all interfaces—PWM DC fan control, ultrasonic ranging (timer capture), keypad decoding, LCD control, and the RS-232 communication to the PC.*

Input variables			Output variable
Current Height	Delta Height	Velocity	Duty Cycle
very lo	neg big	neg big	very slo
lo	neg small	neg med	slo
medium	zero	neg small	medium slo
hi	pos small	zero	medium
very hi	pos big	pos small	medium fast
		pos med	fast
		pos big	very fast

Table 1—*To describe the system adequately, a sufficient number of variables and terms describing the system must be defined.*

Variable	Shell Value min	Shell Value max	Code Value min	Code Value max
Current Height	0	120	0	255
Delta Height	–50	50	0	255
Velocity	–5	5	0	255
Duty Cycle	0	255	0	255

Table 2—*The code value is passed to the fuzzy-logic algorithm and is converted to a shell value within fuzzy logic. The shell value is converted back to a code value when the fuzzy-logic algorithm outputs it.*

This ranging process continues indefinitely until interrupted by the user. The noticeable differences this project has from the PID-pong project, other than the obvious physical ones, are in the control algorithm and the microcontroller.

The control panel houses an ultrasonic ranging module and the microcontroller. The microcontroller handles all of the peripheral interfaces including the keypad, the LCD display, the ultrasonic ranging module, and the RS-232 serial link.

We wanted a microcontroller that could handle the data throughput and all of these peripherals with little or no external components. The best choice for handling all these functions turned out to be Microchip's PIC16C74.

The PIC16C74 contained more than enough on-chip program and data memory. Furthermore, the interrupt capabilities, I/O pins, PWM module, capture and compare modules, timer modules, serial communications interface (SCI), and A/D converter make it a perfect fit for the application.

In addition, the on-chip, pulse-width-modulation (PWM) module allows a single-component (FET) interface for the DC fan control.

The ranging module interfaces directly to the microcontroller. The only external component required is a pull-up resistor on the ECHO line because it is an open-collector output. Also, we replaced the gain resistor (R1) for the receiver on the ultrasonic ranging board with a 20-kΩ potentiometer. This enables us to adjust the gain during debugging to reduce reflections inside the tube.

The other major difference from the PID-pong project is the control algorithm. Not only did we have a much larger project than the ping-pong ball, we had a six-week time constraint. This gave us a month and a half to conceive the project and build it to aesthetically pleasing, trade-show standards.

It was enough of a task getting the hardware assembled in the short time frame, but with a PID control algorithm, the project seemed impossible. So, out of desperation, we thought we would put fuzzy logic to the test. We wanted to see if fuzzy logic would deliver on its promises of accurate control and shorter development time. The development tool we used for fuzzy logic control was Inform Software's fuzzyTECH-MP.

Because the hardware development consumed virtually all of the six-week schedule, there was little time left to develop the control algorithm. We didn't really know how the beach ball would behave in the tube or even if we could reasonably control it.

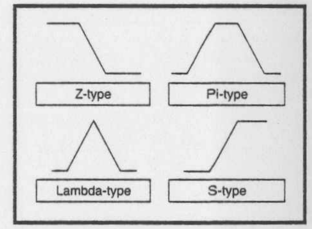


Figure 2—*The standard membership function can be mathematically represented as piecewise linear functions with up to four defining points.*

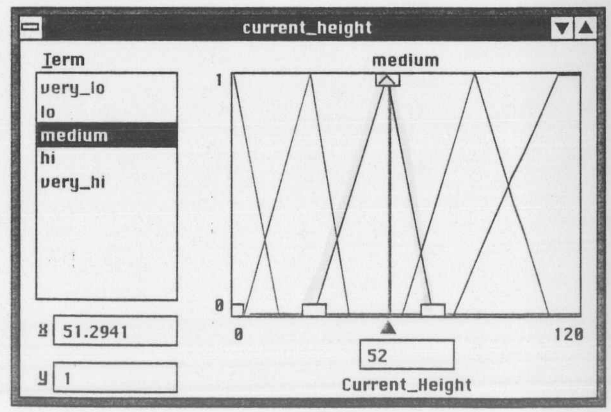
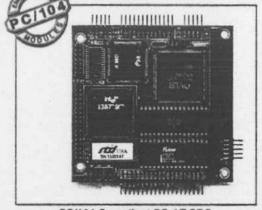


Photo 1—*The term "medium" for the variable Current Height is a Lambda-type membership function centered around 52. When the beach ball has a value of 52 (or 26"), the degree of membership for the beach ball is 1.0 medium. The degree of membership decreases for medium as the beach ball moves in either direction from 52.*

REDUCE THE STACK!
Use fully integrated
PC/104 CPU and DAS
modules from
rtd®

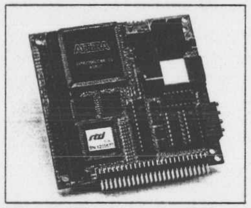
Module size: 90 x 96 x 15mm



PC/104 Compliant PC-AT SBC
CMi386SX-1: $578 /100 pieces
2MB DRAM & SSD software included.

5 PC/XT/AT Single Board cpuModules™:
- 486SLC, 386SX, F8680, V41 & VG230 DOS CPUs
- 80387SX math coprocessor socket on-board
- 512KB, 1MB, 2MB or 4MB DRAM installed
- Two 32-pin SSD sockets & support software
- IDE, floppy & CGA controllers
- RS-232/422/485 serial ports
- Bidirectional parallel, keyboard & speaker ports
- Keypad scanning & PCMCIA support
- Power management & single +5V supply

7 utilityModules™:
- Super VGA controller & I/O modules
- PCMCIA carriers for Types I, II & III cards



PC/104 Compliant 200 kHz Analog I/O Module
DM5408-2: $498 /100 pieces

17 DAS dataModules®:
- 12 & 14-bit A/D conversion up to 200 kHz
- Gap free, high speed sampling under Windows™ & DOS
- Programmable scan, burst & multiburst
- Pre, post & about triggers
- 1K channel-gain scan memory with skip bit
- 1024 sample A/D buffer
- 12-bit analog outputs
- Bit programmable digital I/O with Advanced Digital Interrupt modes
- Incremental encoder interfaces
- 4-20 mA current loop source
- opto-22 compatibility
- Low power & single +5V power supply

For technical specifications and data sheets on PC/104, ISA bus and Eurocard products, call
RTD USA Technical FaxBack: 1 (814) 235-1260
RTD USA BBS: 1 (814) 234-9427

rtd Real Time Devices USA
200 Innovation Boulevard • P.O. Box 906
State College, PA 16804-0906 USA
Tel: 1 (814) 234-8087 • Fax: 1 (814) 234-5218

RTD Europa Fax: (36) 1 212-0260
RTD Scandinavia Fax: (358) 0 346-4539

RTD is a founder of the PC/104 Consortium and the world's leading supplier of PC/104 CPU and DAS modules.

#121

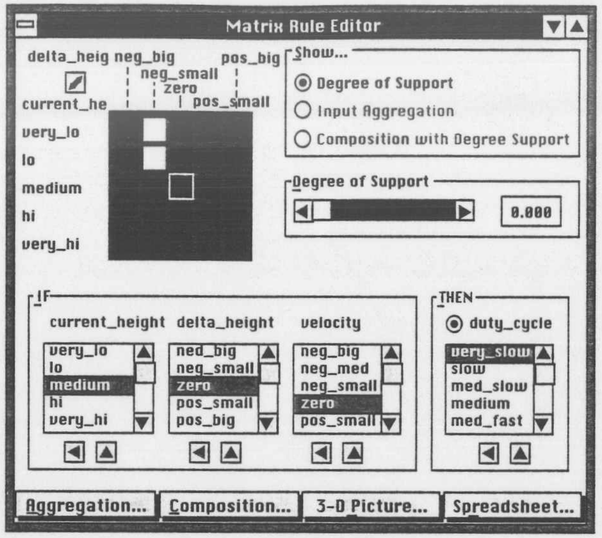


Photo 2—*The Fuzzy Associative Map (FAM) shows the degree of support for each of the rules. For the rule in this example, the degree of support is 0, which indicates a totally implausible rule.*

Finally, five weeks into it, we had the hardware built enough for a manual test. The test was crude, but it did show that control of the beach ball was possible. We at least learned that the algorithm would be able to control the beach ball to within a couple of inches of the desired height.

FUZZY DESIGN

Next, we turned our attention to the fuzzy-logic control algorithm.

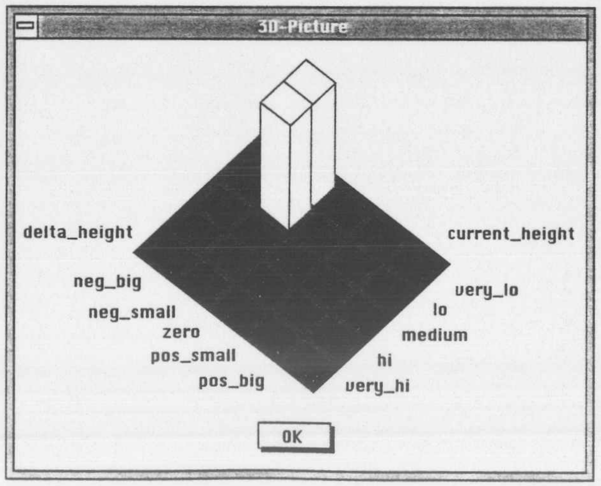


Photo 3—*The rule listed in Photo 2 can be represented as a 3D picture.*

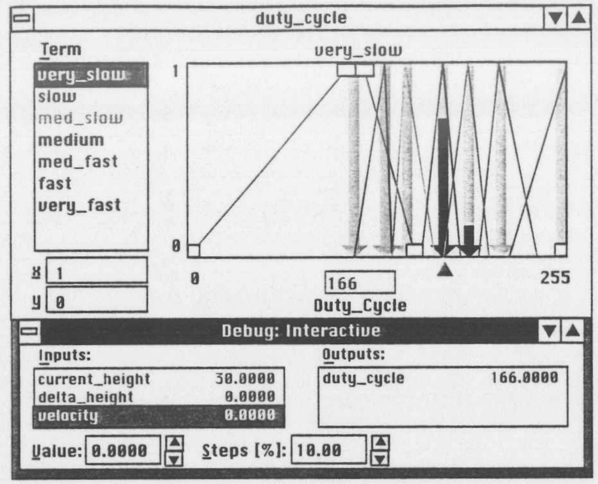


Photo 4—*The crisp value is calculated by an inference weighted mean of the term-membership maxima. That is, the degree of membership for the term medium (long black arrow) is 0.7 and the degree of membership for the term medium fast (short black arrow) is 0.1. The resulting crisp output value is 166.*

Basically, fuzzy logic first translates the crisp inputs from the sensors into a linguistic description. It then evaluates the control strategy contained in fuzzy-logic rules and translates the result back into a crisp value.

Of course, the first step in a fuzzy-logic control design is system definition. This is relatively straightforward for this project. The only possible sources of inputs to the fuzzy-logic control algorithm are the ultrasonic transducer, the user, and the DC fan.

The key is deciding which of these inputs are significant and which aren't. To do this, we put ourselves in the place of the beach ball. We formed a list of critical questions, and for each, we defined a corresponding variable:

- Where am I? → Current Height
- How far am I from where I want to be? → Delta Height
- How fast am I getting there? → Velocity
- What external force will get me there? → Duty Cycle

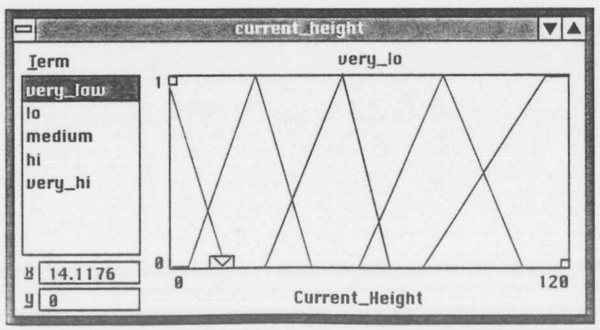


Photo 5—*Once the system-level debugging completes, the final input and output variables are graphically represented. These representations are included in Photos 5–8. Here, although the current height variable contains five terms, we now recognize that three terms would probably have been sufficient. The five terms are fairly symmetrical across the range.*

FREE
Data Acquisition
Catalog

1995 PC and VME data acquisition catalog from the inventors of plug-in data acquisition.

Featuring new low-cost A/D boards optimized for Windows, DSP Data Acquisition, and the latest Windows software.

Plus, informative technical tips and application notes.

Call for your free copy
1-800-648-6589

ADAC
American Data Acquisition Corporation
70 Tower Office Park, Woburn, MA 01801
phone 617-935-3200 fax 617-938-6553

#122

In fuzzy-logic control, the linguistic system definition becomes the control algorithm. And, although defining the variables is the starting point, it isn't good enough to say, "I have velocity." Instead, you need to know to what degree you have velocity.

Determining the extent of a variable is accomplished by defining terms that more fully describe it. The combination of variables and terms gives a linguistic description of what is happening to the system. From this, a variable can be described as having a "positive small velocity" or a "positive big velocity" rather than just a "velocity."

There is no fixed rule on how many terms you need to define a variable. Typically, three to five terms are defined, but more or less may be needed depending on the control algorithm. Table 1 lists the four variables used for the trade-show demo and their associated terms. In retrospect, we probably could have reduced Current Height to three terms and Velocity to five terms.

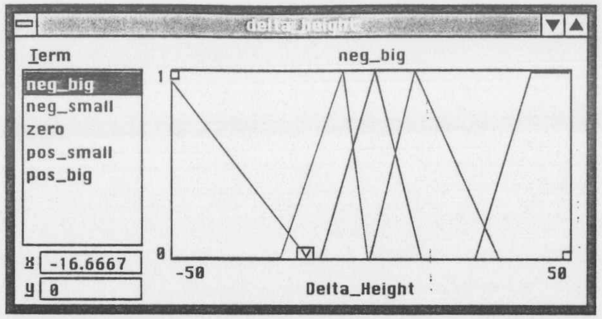


Photo 6—*The delta height variable contains five terms: neg big, neg small, zero, pos small, and pos big. The middle terms bunch together around zero.*

Once the linguistic variables are defined, we start defining data types and values. For this application, we defined data types as 8-bit integers and then specified the shell and code values for each variable. The code value is the crisp number that is used in the digital domain and is used when the code is generated. The shell value is the equivalent number used in the fuzzy domain.

For example, you can define the shell value for Duty Cycle to be a minimum of 0 percent and a maximum of 100. Within the fuzzy-logic development tool, Duty Cycle therefore takes on a value between 0 and 100, inclusive.

Similarly, although the code value is limited by the data type, it can take on any or all of the digital range. That is, if the shell value is 0 to 100, the code values can be 0 to 100. However, to get full resolution, we defined the code values as 0 to 255. The code and shell values are shown in Table 2. Note that for the height and velocity variables, the shell values are scaled by two (e.g., a Current Height with a crisp value of 60 corresponds to 30").

Next, we defined the membership functions that further describe the variables. FuzzyTECH-MP, the fuzzy-logic development tool we used, creates membership functions automatically. Although this gives a good starting point, the membership functions still need to be fine-tuned during debugging. In this application, we used only the linear-shaped functions (Pi, Z, S, and Lambda types) as shown in Figure 2.

FUZZIFICATION

Once the variables are specified, it's time to define the interfaces between the input variables. These interfaces contain the fuzzification procedures, which also need to be defined. For code efficiency, the

Listing 1—*The FTL (Fuzzy Technology Language) code ended up compiling down to 0.7 KB of PIC code and used 29 bytes of data memory.*

```
PROJECT {
  NAME = B_BALL.FTL;
  SHELL = MP;
  COMMENT {
}  /* COMMENT */
  SHELLOPTIONS {
    ONLINE_REFRESHTIME = 55;
    ONLINE_TIMEOUTCOUNT = 0;
    ONLINE_CODE = OFF;
    TRACE_BUFFER = (OFF, PAR(10000));
    BSUM_AGGREGATION = OFF;
    PUBLIC_IO = ON;
    FAST_CMBF = ON;
    FAST_COA  = OFF;
    SCALE_MBF = OFF;
    FILE_CODE = OFF;
    BTYPE = 8_BIT;
  } /* SHELLOPTIONS */
  MODEL {
    VARIABLE_SECTION {
      LVAR {
        NAME    = current_height;
        BASEVAR = Current_Height;
        LVRANGE = MIN(0.000000), MAX(120.000000),
                  MINDEF(0), MAXDEF(255),
                  DEFAULT_OUTPUT(120.000000);
        RESOLUTION = XGRID(0.000000), YGRID(1.000000),
                     SHOWGRID (ON), SNAPTOGRID(ON);
```

(continued)

PC FuzzPong

David Rees-Thomas

For me, fuzzy logic turned Tom Cantrell's PID-Pong into FuzzPong, a fuzzy-logic teaching tool.

The hardware setup is pretty much as Tom Cantrell described in his article (*INK* 42, 50), except that I used a 12-V centrifugal blower instead of a muffin fan. The duty cycle of a pulse-width modulated (PWM) waveform applied to the gate of a power MOSFET determines blower speed. My ultrasonic rangefinder is an old Polaroid demo kit (unmodified) giving 5 samples per second and a minimum range of about 9".

I do the fuzzy calculations on a PC, so I'm able to add a real-time graphics interface to show fuzzy logic in action. The PC screen (Photo I) depicts the outlines of the input and output membership functions (MF) and a map of the control surface generated by the current rule base. The instantaneous values of delta_X (distance from setpoint), dX/dt, and change in controller output appear as moving vertical bars.

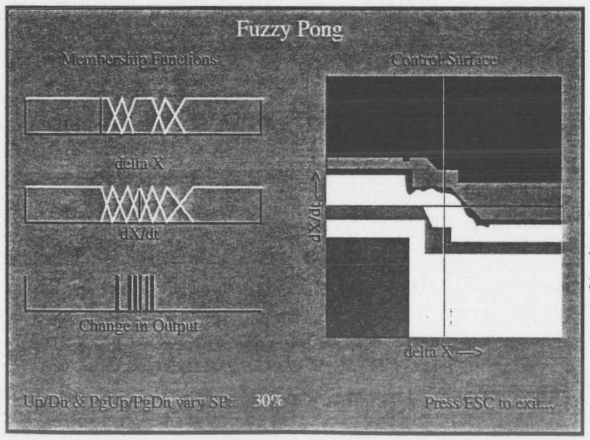


Photo I—*Part of FuzzPong's success as a teaching tool lies in its graphics display. The control surface shows the controller output for all possible values of the two input variables. Red indicates areas of large positive change in blower speed, while blue depicts regions of large negative change. The white region indicates little or no change in blower speed.*

An MC68HC11 E9 microcontroller does the low-end measurement and control work, communicating with the PC serially at 9600 bps. Input Captures monitor two signals on the rangefinder logic board to give a 16-bit value proportional to the height of the ball. This value is transmitted to the PC as four ASCII characters.

The control value returned by the PC is 8-bit binary and represents a change in the duty cycle of the PWM waveform. A toggle switch selects fuzzy or manual control. In manual mode, a 2-kΩ pot, connected to one of the 'HC11's ADC inputs, sets the PWM duty cycle.

FuzzPong is written in Turbo C and takes advantage of that com-

Listing I—*This code fuzzifies crisp inputs, performs max-min composition, and defuzzifies weighted output membership functions to yield crisp output value.*

```
/* F_CONTROL-fuzzy computation on crisp inputs T, Tdot */
unsigned int f_control(unsigned int T, unsigned int Tdot)
{
  int i, j;
  float f_T[9], f_Tdot[9];            /* input membership value */
  float f_out[9];                     /* output singleton MFs */
  unsigned int output;                /* crisp output vale */
  for (j = 0; j < N_T; j++)           /* fuzzification-m_T[] */
    f_T[j] = fuzz (T, m_T[j]);        /* and m_Tdot are input */
                                      /* membership functions */
  for (i = 0; i < N_Tdot; i++)        /* defined in file */
    f_Tdot[i] = fuzz (Tdot, m_Tdot[i]);   /* FUZZYSET.DAT */
  infer(rule, f_Tdot, f_T, f_out0;  /* MAX-MIN composition */

  output = defuzz(f_out, m_OUT, N_OUT); /* defuzzification */
  return (output);
}

/* FUZZ-returns membership of input in a fuzzy set */
float fuzz(unsigned int input, unsigned int fm[4])
{
  if ((input >= fm[1]) && (input <= fm[2]))  /* fm[] is MF */
    return (1.0);                            /* definition */
  else if ((input > fm[0]) && (input < fm[1]))
    return ((float)(input - fm[0])/(float)(fm[1] - fm[0]));
  else if ((input > fm[2]) && (input < fm[3]))
    return ((float)(fm[3] - input)/(float)(fm[3] - fm[2]));
  else return (0.0);
}
```

(continued)

›utation of fuzzification is carried t runtime.
n this project, the type of fication used is a membership-ion computation. This choice is ›y due to the code-space efficiency ccuracy of this method. Once fication has taken place, the ithm is performed in the fuzzy l according to the rule base.

ZY RULE BASE

Next, we are ready for fuzzy ence. The entire fuzzy inference is ined within the rule blocks of a m. For example, if the beach ball ır the top of the tube and we nanded it to be near the bottom of ıbe, the rule that describes the tion would be:

rrent Height = very hi
ND Delta Height = neg big
N Duty Cycle = slow

definition continues until we adequately described the system. that the IF part of the fuzzy

Listing 1—*continued*

```
TERM {
  TERMNAME = very_lo;
  POINTS = (0.000000, 1.000000),
           (14.117647, 0.000000),
           (120.000000, 0.000000);
  SHAPE = LINEAR;
  COLOR = RED (255), GREEN (0), BLUE (0);
}
TERM {
  TERMNAME = lo;
  POINTS = (0.000000, 0.000000),
           (5.176471, 0.000000),
           (24.941176, 1.000000),
           (40.941176, 0.000000),
           (120.000000, 0.000000);
  SHAPE = LINEAR;
  COLOR = RED (0), GREEN (255), BLUE (0);
}
TERM {
  TERMNAME = medium;
  POINTS = (0.000000, 0.000000),
           (27.294118, 0.000000),
           (51.294118, 1.000000),
           (66.352941, 0.000000),
           (120.000000, 0.000000);
  SHAPE = LINEAR;
  COLOR = RED (0), GREEN (0), BLUE (255);
}
TERM {
```

(continued)

Listing I—*continued*

```
/* INFER—max-min composition on crisp inputs T, Tdot */
void infer(int rule[][9], float f_Tdot[], float fT[],
           float f_out[])
{
  int i, j, k;
  float cons[9][9];                /* weights of rule o/ps */

  for (i = 0; i < N_Tdot; i++)    /* compute min for each */
    for (j = 0; j < N_T; j++)     /* combination of inputs */
      cons[i][j] = amin (f_Tdot[i], f_T[j]);

  for (k = 0; k < N_OUT; k++)     /* clear fuzzy o/p array */
    f_out[k] = 0.0;

  for (i = 0; i < N_Tdot; i++)    /* compute max for each */
    for (j = 0; j < N_T; j++) {   /* output membership fcn */
      k = rule[i][j];
      if (f_out[k] < cons[i][j])  /* giving fuzzy weight */
        f_out[k] = cons[i][j];    /* for each output MF */
    }
}

/* DEFUZZ- COG defuzzification of singletons MFs */
unsigned int defuzz(float f_out[], unsigned int m_out[],
                    int n)
{
  int i;                          /* f_out[i] is the fuzzy */
  float crisp   = 0;              /* weight computed for */
  float weights = 0;              /* the singleton output */
                                  /* MF m_out[i] */
```

(continued)

piler's graphics library. The program includes three main modules: FUZZMAIN, which contains the graphics routines, FUZZCOMM, which handles data to and from the 'HC11, and FUZZMATH, which is the actual fuzzy controller.

In addition to managing the graphics display, FUZZMAIN runs the executive loop, which keeps the whole show going. FUZZCOMM scales height values received from the 'HC11 and computes their rate of change using a three-point, backward-difference formula. It also massages output data prior to transmission to the microcontroller. Here, I experiment with various software filters and smoothing algorithms, with dubious results.

FUZZMATH (see Listing 1) does its fuzzification, inference, and defuzzification straightforwardly. I stuck to trapezoidal or triangular membership functions for the input fuzzy sets and singletons for the outputs. FUZZMATH performs the usual max-min composition to

Listing 1—*continued*

```
      TERMNAME = hi;
      POINTS = (0.000000, 0.000000),
               (55.529412, 0.000000),
               (82.352941, 1.000000),
               (106.352941, 0.000000),
               (120.000000, 0.000000);
      SHAPE = LINEAR;
      COLOR = RED (128), GREEN (0), BLUE (0);
    }
    TERM {
      TERMNAME = very_hi;
      POINTS = (0.000000, 0.000000),
               (73.411765, 0.000000),
               (113.411765, 1.000000),
               (120.000000, 1.000000);
      SHAPE = LINEAR;
      COLOR = RED (0), GREEN (128), BLUE (0);
    }
  } /* LVAR */
.
.
.
} /* VARIABLE_SECTION */

  OBJECT_SECTION {
    INTERFACE {
      INPUT = (current_height, FCMBF);
      POS = -213, -137;
      RANGECHECK = ON;
```

(continued)

inference is aggregation and can be AND or OR.

The rules of the rule block can be defined in terms of plausibility. A plausible rule is defined by a 1.0 while a totally implausible rule is defined by 0.0. The degree to which a crisp value belongs to a term is known as the *degree of membership*.

For example, the terms *medium* and *hi* for the variable Current Height are defined as a Lambda-type membership function centered around the crisp values 52 (26") and 82 (41"), respectively, as shown in Photo 1.

Therefore, if the beach ball was at 26", the degree of membership is 1.0 for medium and 0.0 for hi. However, as the beach ball rises in height, the degree of membership for the term medium decreases and the degree of membership for hi increases.

The interplay of these linguistic variable terms is controlled by the rule base, which defines not only the relationship between the terms, but also how much each rule is supported. The support of a rule, or plausibility, is

generate fuzzy outputs, combining them to produce a crisp output value by center-of-gravity weighting.

FuzzPong uses membership functions and a rule base, defined in an ASCII text file (Listing II). It's easy to change the number and limits of membership functions or to tweak the rules so you can see the effect of the changes. Even without a real pong system connected, FuzzPong's control surface shows roughly how the controller reacts in each case. (Note: the surface shows controller action only and *not* overall system response!)

HOW WELL DOES IT WORK?

I haven't made any quantitative measurements, but in the absence of external disturbance, FuzzPong can hold the ball within roughly one ball diameter of the setpoint. It recovers nicely if the system is "bumped" by placing a finger across the end of the tube. Both bumping and a change of setpoint show a

Listing I—*continued*

```
  for (i = 0; i < n; i++){
    crisp += f_out[i] * (float) m_out[i];
    weights += f_out[i];       /* compute weighted average */
  }
  return (unsigned int)(crisp/weights);
}
```

Listing II—*FUZZYSET.DAT includes fuzzy membership functions and rules.*

```
* Input MFs are entered as four hex values A B C D
* where the MF has the generalized trapezoidal shape:
*              B---C
*             /     \
*            /       \
*           A         D
* N_T (number of MF for first input variable):
  5
* Membership function names:
    NL NS ZR PS PL
* Membership function limits:
    0x00 0x00 0x60 0x70
    0x60 0x70 0x70 0x7C
    0x70 0x7C 0x88 0x98
    0x88 0x98 0x98 0xB0
    0x98 0xB0 0xFF 0xFF

* N_Tdot (number of MF for second input variable):
  5
```

(continued)

known as the *degree of support* for that rule.

From the list of rules, a Fuzzy Associative Map (FAM) is constructed. As you can see in Photos 2 and 3, the FAM shows the plausibility (degree of support) of each rule.

DEFUZZIFICATION

The interface for the output variables contains the defuzzification procedures. This project, like most control applications, the center-of-maximum (CoM) method is used for defuzzification.

CoM evaluates multiple output term as valid and makes a compromise between them by computing a weighted mean of the term-membership maxima. The example in Photo 4 shows defuzzification of the linguistic variable Duty Cycle using CoM.

The crisp values of the three input variables used in Photo 4 are:

Current Height:	30
Delta Height:	0
Velocity:	0

Listing 1—*continued*

```
  }
  INTERFACE {
    INPUT = (delta_height, FCMBF);
    POS = -216, -83;
    RANGECHECK = ON;
  }
  INTERFACE {
    OUTPUT = (duty_cycle, COM);
    POS = 158, -79;
    RANGECHECK = ON;
  }
  RULEBLOCK {
    INPUT = current_height, delta_height, velocity;
    OUTPUT = duty_cycle;
    AGGREGATION = (MIN_MAX, PAR (0.000000));
    COMPOSITION = (GAMMA, PAR (0.000000));
    POS = -39, -113;
    RULES {
      IF    current_height = very_lo
        AND delta_height = neg_big
      THEN  duty_cycle = slow   WITH 1.000;
      IF    current_height = very_lo
        AND delta_height = neg_small
      THEN  duty_cycle = med_slow   WITH 1.000;
      .
      .
      .
      IF current_height = very_hi
        AND delta_height = pos_small
```

(continued)

Listing II—*continued*

```
* Membership function names:
    NL NM NS ZR PS PM PL

* Membership function limits:
    0x00 0x00 0x50 0x70
    0x50 0x70 0x70 0x7C
    0x70 0x7C 0x72 0x98
    0x82 0x98 0x98 0xC0
    0x98 0xC0 0xFF 0xFF

* N_OUT (number of output MFs):
    5
* Singleton output function (0x80 => zero change):
*     NL   NS   ZR   PS   PL
    0x70 0x7C 0x80 0x82 0x88

* Fuzzy rule base (FAM matrix): the consequent of each rule
* is the index of the corresponding output MF,e.g., 2 => ZR
* Tdot  T -> NL NS ZR PS PL
* NL
            4  3  3  3  3
* NS
            4  4  3  2  2
* ZR
            3  3  2  1  1
* PS
            2  1  1  0  0
* PL
            0  0  0  0  0
```

fairly heavily damped response. FuzzPong also handles a ball wrapped with one turn of electrical tape without significant loss of control.

All in all, the exercise of writing and using FuzzPong has been a great introduction to fuzzy control. ■

David Rees-Thomas has a B.Sc. in chemistry and math from Queen's University and a diploma in Electronics Technology from Northern College in Kirkland Lake, Ontario. For the last ten years, he has been teaching at the British Columbia Institute of Technology in Burnaby, BC, where he specializes in microcontrollers and data communications. David may be reached at resd2215@bcit.bc.ca.

IRS

407 Very Useful
408 Moderately Useful
409 Not Useful

Listing 1—*continued*

```
        AND velocity = neg_big
      THEN duty_cycle = very_fast with 1.000;
    } /* RULES */
  }
  INTERFACE {
    INPUT = (velocity, FCMBF);
    POS = -211, -29;
    RANGECHECK = ON;
  }
  } /* OBJECT_SECTION */
 } /* MODEL */
} /* PROJECT */
TERMINAL {
    BAUDRATE     = 9600;
    STOPBITS     = 1;
    PROTOCOL     = NO;
    CONNECTION   = PORT1;
    INPUTBUFFER  = 4096;
    OUTPUTBUFFER = 1024;
} /* TERMINAL */
```

The crisp value can be calculated using the CoM method with the following equation:

$$C = \frac{\Sigma i[I \times \max_x(M) \times \arg(\max_x(M))]}{\Sigma i I} = \frac{(0.7 \times 165) + (0.1 \times 178)}{0.7 + 0.1} = 166$$

where *C* is the crisp output value, *i* is the linguistic term, *I* is the inference result, and *M* is the membership function of the linguistic term.

For this example, when the crisp values are fuzzified, the Duty Cycle variable is defined to be mostly medium (≈0.7 degree of membership) and somewhat medium fast (≈0.1 degree of membership). The arguments for the medium and medium-fast term membership maxima are 165 and 178, respectively. When this fuzzy description is defuzzified, the output is the crisp value 166 as is shown in Photo 4.

SHOW TIME

The first time we ran the demo, the beach ball barely lifted off the DC fan. Apparently, we had our Duty Cycle defined too low. So, in real time, we shifted the Duty Cycle terms to the right and watched the beach ball slowly lift off the DC fan. We adjusted the Duty Cycle so that the beach ball reached 30″. We played with the Delta Height terms—we bunched neg small, zero, and pos small—and the beach ball stabilized at 30″. There was virtually no fluctuation in the height.

Although 30″ was a good starting point, we knew that the system was highly nonlinear. So, we began testing the system at extreme levels and moving the beach ball at different rates from one extreme to the other.

From the manual control tests performed earlier, we had a good characterization of how the beach ball would behave in the extreme regions. It turned out that terms for Current Height and Velocity needed almost no adjustment. In fact, the Velocity variable was not even used.

The variable that required the most work was the Duty Cycle. But before the end of the day, the algorithm was working well beyond our expectations. The beach ball could go from resting, with the DC fan off, to the maximum allowable height of 42″ in less than 8 s with no overshoot. Operation between the minimum and maximum height was much quicker, and there was no overshoot.

We felt confident that we could sleep well that night. Ironically, it was the last sleep we got for a while. During the night, a cold front moved in. When we tried to run the beach ball demo the next day, it sent the beach ball to the top of the tube every time.

To make a long story short, the problem turned out to be with the ranging module. The receiver gain was set a little too high. The potentiometer was set just below the level of receiving reflections in the tube. The changes in the environment pushed it over the edge. After a minor adjustment to the potentiometer, we were up and running again.

However, this time, once we started the demo again, the beach ball would stop 6″ short of the desired height. After thinking about what else we may have missed, the answer hit us like a blast of cold air—literally.

The cold front changed the atmospheric conditions enough so that the DC fan didn't have enough juice to push the ball up to the desired height. This is where Velocity, our one unused term, came into play. We decided to

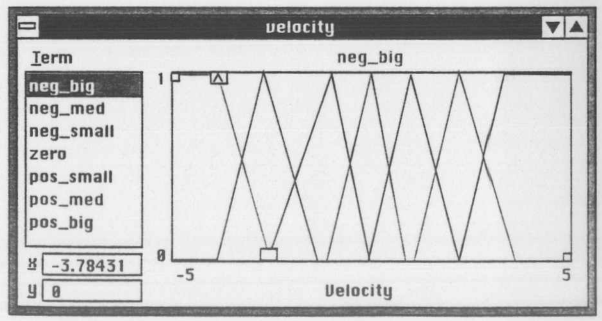


Photo 7—*The velocity variable contains seven terms: neg big, neg med, neg small, zero, pos small, pos med, and pos big. The terms are nearly symmetrical across the range. With hindsight, we realize that these seven terms could be reduced to five.*

add a few rules that used Velocity to nudge the ball into place—you know, as sort of a turbo mode. With this adjustment, the demo worked.

Photos 5, 6, 7, and 8 graphically depict the final state of the linguistic variables. Listing 1 offers an excerpt of the Fuzzy Technology Language (FTL) that we used. (FTL is a vendor and hardware-independent language which defines the fuzzy-logic based system.)

Once we had completed the fuzzy logic algorithm, we ran the assembler to get an estimate of the memory needed to embed it in the PIC16C74. The fuzzy logic algorithm used approximately 0.7 KB of program memory and 41 bytes of data memory. The total code space for the project was 1 KB of program memory and 80 bytes of data memory. Including the fuzzy logic algorithm, we still had well over 50% of the memory resources available on the PIC16C74.

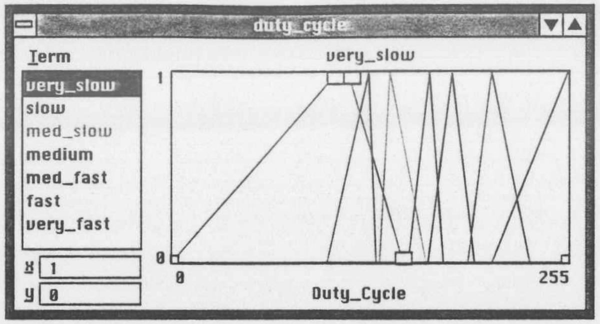


Photo 8—*The duty cycle variable contains seven terms: very slow, slow, med slow, medium, med fast, fast, and very fast. The terms bunch together around medium.*

FUZZY CHALLENGE

Our trade show demo was very successful. The positive feedback virtually guaranteed that the demo will surface again at future trade shows. However, now that the public has seen the demo, marketing wants to capitalize on its success by adding enhancements.

Two enhancements are already in the works. The first includes adding manual control to allow a user to challenge the fuzzy logic control. The second entails breaking the serial communication link and embedding the fuzzy logic in the microcontroller.

Finally, if we get crazy enough, we'll remove the tube and run the demo in free air.

So, if you happen to see us at a trade show near you, come put fuzzy logic to the challenge! ■

Special thanks to John Day, Rodney Duke, and Mort Simmonds for their help with the project.

Robert Schreiber is a senior applications engineer at Microchip Technology. He has more than ten years of embedded systems, hardware, and software design experience. He may be reached at apps@mchp.com.

SOURCES

Microchip Technology, Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224
(602) 786-7200
Fax: (602) 899-9210

Inform Software, Inc.
1840 Oak Ave.
Evanston, IL 60201
(708) 866-1838
Fax: (708) 866-1839

IRS

410 Very Useful
411 Moderately Useful
412 Not Useful

Q. How do you know you're getting the most from your development tool purchase?

A. Compare Avocet Systems with the competition.

- A Broad Line of High-Quality Products at Competitive Prices
- Free On-Line Technical Support for Registered Users. No Voicemail!
- Attractive Multi-User Discount Prices & Our "50%+" Educational Discount Plan
- Unconditional 30-Day Money-Back-Guarantee

ANSI-C COMPILERS
8051 • 68xxx • 68HC11
6805 • Z80/180/64180
6801 • 6809
H8/300 & More

MACRO ASSEMBLERS
8051 • 68xxx • 68HC11 6805
• Z80/180 • Z8 • 8096/196
6800/6801 • 6809
H8/300 • 8048 & More

SIMULATORS/DEBUGGERS
8051 • 680xx • 68HC11 6805
• Z80/180 • 6800
6809 • 8048 • 6502
8085 & More

AND HARDWARE
EPROM Programmers & Emulators by EETools, Tribal, Softaid, Cactus Logic

Now call the obvious choice!

AVOCET SYSTEMS, INC.
The Best Source for Quality Embedded System Tools
(800) 448-8500
(207) 236-9055 Fax (207) 236-6713

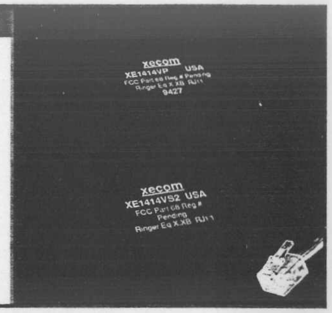
#123

Embedded Modems, DAAs & Controllers

Easy to Use — Plug and Play Modules

Xecom Component Modem Modules

	Data Rate	Host I/F	Small Size
XE1212BL5	1200 bps	8250B UART	2.28"×1.0"×0.5"
XE1276	1200 bps	Serial	2.0"×1.125"×0.44"
XE2401	2400 bps	Serial	2.28"×1.08"×0.42"
XE2401P	2400 bps	8250B UART	2.28"×1.08"×0.42"
XE2476	2400 bps	Serial	2.0"×1.125"×0.44"
XE9601	9600 bps	Serial	2.28"×1.08"×0.42"
XE9624GS	2400 bps	Serial	1.52"×1.52"×0.42"
XE1414VP	14,400 bps	16C450 UART	2.75"×1.38"×0.42"
XE1414VS	14,400 bps	Serial	1.52"×1.52"×0.42"



Xecom modems are complete modems not just modem chips. They have the telephone interface and all are FCC Part 68 registered. Use our registration, add a telephone jack, and connect directly to the telephone network.

Xecom Component DAA Modules

	Data Rate	Useage	Small Size
XE0002B	to 9600	North America	1.25"×1.0"×0.50"
XE0017	to 28,800	North America	1.0"×1.0"×0.40"
XE0054SIP	to 14,400	North America	1.5"×0.5"×0.50"
XE0058SIP	to 28,800	North America	1.5"×0.5"×0.30"
XE0068	to 14,400	North America	1.75"×1.08"×0.42"
XE1030	to 28,800	UK, other EU	1.6"×0.5"×0.45"
XE1040	to 28,800	Germany	1.6"×0.5"×0.45"
XE1050	to 28,800	France	1.6"×0.5"×0.45"

Xecom's DAA modules provide all the circuitry to connect your equipment to the telephone network. You don't have to add any relays or transformers to complete the circuit.

Introducing Xecom Component Controller Modules

BLACKJACK TELE-CONTROLLER

XE5200
- 80C552 microprocessor
- 128 KByte Flash Memory
- 26 I/O Lines
- Embedded design tools
- Small size: 2.75"×1.38"×0.50"

XE5224DM
- XE5200 plus 2400 bps modem

XE5224FD
- XE5224DM plus fax send & receive

DOMINO CONTROLLER

XE5052
- 80C52 microprocessor
- 32 KByte EEPROM
- 32 KByte RAM
- Embedded full floating-point Basic
- Small size: 1.75"×1.06"×0.40"

XE5052A
- XE5052 plus 12-bit A/D convertor

Xecom's controller modules contain the complete controller: processor, memory, and embedded design tools. Add your applications program and you're on-line.

xecom Embedded Modem, Controller and DAA modules

374 Turquoise Street, Milpitas, CA 95035 Phone: 408-945-6640 Fax: 408-942-1346 Email: info@xecom.com

#124